Greedy Algorithms in Compressed Sensing

Jeff Blanchard

Minitutorial: Compressed Sensing/Dimension Reduction
SIAM Annual Meeting 2017, Pittsburgh, July 14, 2017

Compressed Sensing

Measure and recover a s-sparse vector with an $m \times n$ matrix:

- The problem is characterized by three parameters: $s < m < n$
  - n, the signal length;
  - m, number of (inner product) measurements;
  - s, the sparsity of the signal.
- The sampling/sensing matrix $\mathcal{A}$ is of size $m \times n$.
- The signal $f \in \mathbb{R}^n$ is s-sparse in some sense, $f = Dx$ with $\|x\|_0 = s$.
  - We'll simplify a few things by assuming $D = I$ so that $f = x$.

Compressed Sensing

Measure and recover a s-sparse vector with an $m \times n$ matrix:

- The problem is characterized by three parameters: $s < m < n$
  - n, the signal length;
  - m, number of (inner product) measurements;
  - s, the sparsity of the signal.
- The sampling/sensing matrix $\mathcal{A}$ is of size $m \times n$.
- The signal $f \in \mathbb{R}^n$ is s-sparse in some sense, $f = Dx$ with $\|x\|_0 = s$.
  - We'll simplify a few things by assuming $D = I$ so that $f = x$.

Compressed Sensing

Measure and recover a s-sparse vector with an $m \times n$ matrix:

- The problem is characterized by three parameters: $s < m < n$
  - n, the signal length;
  - m, number of (inner product) measurements;
  - s, the sparsity of the signal.
- The sampling/sensing matrix $\mathcal{A}$ is of size $m \times n$.
- The signal $f \in \mathbb{R}^n$ is s-sparse in some sense, $f = Dx$ with $\|x\|_0 = s$.
  - We'll simplify a few things by assuming $D = I$ so that $f = x$.

Compressed Sensing

Measure and recover a s-sparse vector with an $m \times n$ matrix:

- The problem is characterized by three parameters: $s < m < n$
  - n, the signal length;
  - m, number of (inner product) measurements;
  - s, the sparsity of the signal.
- The sampling/sensing matrix $\mathcal{A}$ is of size $m \times n$.
- The signal $x \in \mathbb{R}^n$ is s-sparse, $\|x\|_0 = s$.
- The samples/measurements $y \in \mathbb{R}^m$ where $y = \mathcal{A}x + \xi$.

Compressed Sensing

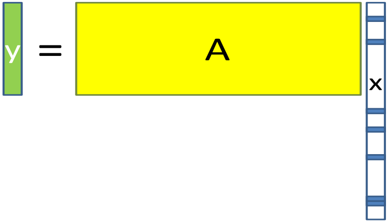
Measure and recover a s-sparse vector with an $m \times n$ matrix:

- The problem is characterized by three parameters: $s < m < n$
  - n, the signal length;
  - m, number of (inner product) measurements;
  - s, the sparsity of the signal.
- The sampling/sensing matrix $\mathcal{A}$ is of size $m \times n$.
- The signal $x \in \mathbb{R}^n$ is s-sparse, $\|x\|_0 = s$.
- The samples/measurements $y \in \mathbb{R}^m$ where $y = \mathcal{A}x + \xi$.

Compressed Sensing

Measure and recover a s-sparse vector with an $m \times n$ matrix:

- The problem is characterized by three parameters: $s < m < n$
  - n, the signal length;
  - m, number of (inner product) measurements;
  - s, the sparsity of the signal.
- The sampling/sensing matrix $\mathcal{A}$ is of size $m \times n$.
- The signal $x \in \mathbb{R}^n$ is s-sparse, $\|x\|_0 = s$.
- The samples/measurements $y \in \mathbb{R}^m$ where $y = \mathcal{A}x$.



Convex Relaxations

Replace the combinatorial optimization problem with its convex envelop.

- Compressed Sensing: ℓ_1-minimization

$$\min_{z \in \mathbb{R}^n} \|z\|_1 \text{ subject to } \mathcal{A}x = y = \mathcal{A}z$$

- Matrix Completion: nuclear norm minimization

$$\min_{Z \in \mathbb{R}^{m \times n}} \|Z\|_* \text{ subject to } \mathcal{A}(X) = y = \mathcal{A}(Z)$$

(where $\|\cdot\|_*$ is the ℓ_1 norm of the singular values)

Many algorithms to solve these optimization problems.

Convex Relaxations

Replace the combinatorial optimization problem with its convex envelop.

- Compressed Sensing: ℓ_1-minimization

$$\min_{z \in \mathbb{R}^n} \|z\|_1 \text{ subject to } \mathcal{A}x = y = \mathcal{A}z$$

- Matrix Completion: nuclear norm minimization

$$\min_{Z \in \mathbb{R}^{m \times n}} \|Z\|_* \text{ subject to } \mathcal{A}(X) = y = \mathcal{A}(Z)$$

(where $\|\cdot\|_*$ is the ℓ_1 norm of the singular values)

Many algorithms to solve these optimization problems.

Greedy Algorithms

Directly solve the combinatorial optimization problem by iteratively searching for the correct low dimensional model.

- Basic: OMP, Hard Thresholding
- Iterative Hard Thresholding: IHT, NIHT (Blumensath & Davies), CGIHT (B., Tanner, Wei)
- Two Stage Pursuits:
  - CoSaMP: Compressive Sampling Matching Pursuit (Needell & Tropp)
  - SP: Subspace Pursuit (Dai & Milenkovic)
  - HTP: Hard Thresholding Pursuit (Foucart, Maleki, Blumensath)

- These all have sufficient conditions for guaranteed uniform recovery.
- These all have variants for row-sparse approximation.
- These all have variants for matrix completion.

Greedy Algorithms

Directly solve the combinatorial optimization problem by iteratively searching for the correct low dimensional model.

- Basic: OMP, Hard Thresholding
- Iterative Hard Thresholding: IHT, NIHT (Blumensath & Davies), CGIHT (B., Tanner, Wei)
- Two Stage Pursuits:
  - CoSaMP: Compressive Sampling Matching Pursuit (Needell & Tropp)
  - SP: Subspace Pursuit (Dai & Milenkovic)
  - HTP: Hard Thresholding Pursuit (Foucart, Maleki, Blumensath)

- These all have sufficient conditions for guaranteed uniform recovery.
- These all have variants for row-sparse approximation.
- These all have variants for matrix completion.

Greedy Algorithms

Directly solve the combinatorial optimization problem by iteratively searching for the correct low dimensional model.

- Basic: OMP, Hard Thresholding
- Iterative Hard Thresholding: IHT, NIHT (Blumensath & Davies), CGIHT (B., Tanner, Wei)
- Two Stage Pursuits:
  - CoSaMP: Compressive Sampling Matching Pursuit (Needell & Tropp)
  - SP: Subspace Pursuit (Dai & Milenkovic)
  - HTP: Hard Thresholding Pursuit (Foucart, Maleki, Blumensath)
- These all have sufficient conditions for guaranteed uniform recovery.
- These all have variants for row-sparse approximation.
- These all have variants for matrix completion.

Greedy Algorithms

Directly solve the combinatorial optimization problem by iteratively searching for the correct low dimensional model.

- Basic: OMP, Hard Thresholding
- Iterative Hard Thresholding: IHT, NIHT (Blumensath & Davies), CGIHT (B., Tanner, Wei)
- Two Stage Pursuits:
  - CoSaMP: Compressive Sampling Matching Pursuit (Needell & Tropp)
  - SP: Subspace Pursuit (Dai & Milenkovic)
  - HTP: Hard Thresholding Pursuit (Foucart, Maleki, Blumensath)

- These all have sufficient conditions for guaranteed uniform recovery.
- These all have variants for row-sparse approximation.
- These all have variants for matrix completion.

Greedy Algorithms

Directly solve the combinatorial optimization problem by iteratively searching for the correct low dimensional model.

- Basic: OMP, Hard Thresholding
- Iterative Hard Thresholding: IHT, NIHT (Blumensath & Davies), CGIHT (B., Tanner, Wei)
- Two Stage Pursuits:
  - CoSaMP: Compressive Sampling Matching Pursuit (Needell & Tropp)
  - SP: Subspace Pursuit (Dai & Milenkovic)
  - HTP: Hard Thresholding Pursuit (Foucart, Maleki, Blumensath)

- These all have sufficient conditions for guaranteed uniform recovery.
- These all have variants for row-sparse approximation.
- These all have variants for matrix completion.

OMP

Greedy Algorithm to iteratively build the support set one index at a time.

Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support of x_{j-1}

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the residual
- $i_j = \arg\max(|r_j|)$ column index with greatest correlation to the residual
- $T_j = T_{j-1} \cup \{i_j\}$ updated approximate support set
- $x_j = \arg\min_{\text{supp}(z)\subset T_j} \|y - \mathcal{A}z\|_2$ optimal approximation supported on T_j

OMP

Greedy Algorithm to iteratively build the support set one index at a time.

Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support of x_{j-1}

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the residual
- $i_j = \arg\max(|r_j|)$ column index with greatest correlation to the residual
- $T_j = T_{j-1} \cup \{i_j\}$ updated approximate support set
- $x_j = \arg\min_{\text{supp}(z) \subset T_j} \|y - \mathcal{A}z\|_2$ optimal approximation supported on T_j

OMP

Greedy Algorithm to iteratively build the support set one index at a time.

Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support of x_{j-1}

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the residual
- $i_j = \arg\max(|r_j|)$ column index with greatest correlation to the residual
- $T_j = T_{j-1} \cup \{i_j\}$ updated approximate support set
- $x_j = \underset{\text{supp}(z)\subset T_j}{\arg\min} \|y - \mathcal{A}z\|_2$ optimal approximation supported on T_j

OMP

Greedy Algorithm to iteratively build the support set one index at a time.

Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support of x_{j-1}

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the residual
- $i_j = \arg\max(|r_j|)$ column index with greatest correlation to the residual
- $T_j = T_{j-1} \cup \{i_j\}$ updated approximate support set
- $x_j = \underset{\text{supp}(z)\subset T_j}{\arg\min} \|y - \mathcal{A}z\|_2$ optimal approximation supported on T_j

OMP

Greedy Algorithm to iteratively build the support set one index at a time.

Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support of x_{j-1}

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the residual
- $i_j = \arg\max(|r_j|)$ column index with greatest correlation to the residual
- $T_j = T_{j-1} \cup \{i_j\}$ updated approximate support set
- $x_j = \underset{\text{supp}(z)\subset T_j}{\arg\min} \|y - \mathcal{A}z\|_2$ optimal approximation supported on T_j

OMP Demo

Hard Thresholding: As an algorithm

Select the approximation support all at once and project.

- $T = \texttt{PrincipalSupport}_s(\mathcal{A}^* y)$ the support set of the s largest magnitude entries in $\mathcal{A}^* y$
- $x = \arg\min_{\text{supp}(z) \subset T} \|y - \mathcal{A}z\|_2$ optimal approximation supported on T and zeros all other entries.

Hard Thresholding: As an algorithm

Select the approximation support all at once and project.

- $T = \texttt{PrincipalSupport}_s(\mathcal{A}^* y)$ the support set of the s largest magnitude entries in $\mathcal{A}^* y$
- $x = \arg\min_{\text{supp}(z) \subset T} \|y - \mathcal{A}z\|_2$ optimal approximation supported on T and zeros all other entries.

Hard Thresholding: As an initialization

The remaining algorithms all initialize with a hard threshold.

- $T = \texttt{PrincipalSupport}_s(\mathcal{A}^* y)$ the support set of the s largest magnitude entries in $\mathcal{A}^* y$
- $x = \texttt{Threshold}(\mathcal{A}^* y, T)$ optimal approximation supported on T and zeros all other entries.

IHT

A gradient descent with evolving support.

Previous Iteration

- x_{j-1} a s-sparse approximation

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $w_j = x_{j-1} + r_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

IHT

A gradient descent with evolving support.

Previous Iteration

- x_{j-1} a s-sparse approximation

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $w_j = x_{j-1} + r_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

IHT

A gradient descent with evolving support.

Previous Iteration

- x_{j-1} a s-sparse approximation

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $w_j = x_{j-1} + r_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

IHT

A gradient descent with evolving support.

Previous Iteration

- x_{j-1} a s-sparse approximation

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $w_j = x_{j-1} + r_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

IHT

A gradient descent with evolving support.

Previous Iteration

- x_{j-1} a s-sparse approximation

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $w_j = x_{j-1} + r_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

NIHT

A subspace restricted steepest descent with evolving support.

Previous Iteration

- x_{j-1} a s-sparse approximation

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- α_j a near optimal step size
- $w_j = x_{j-1} + \alpha_j r_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

HTP

A variant replacing thresholding with an optimal approximation.

Previous Iteration

- x_{j-1} a s-sparse approximation

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- α_j a near optimal step size
- $w_j = x_{j-1} + \alpha_j r_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \arg\min_{\text{supp}(z) \subset T_j} \|y - \mathcal{A}z\|_2$ optimal approximation supported on T_j

CoSaMP

A different, two-stage approach.

Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support set of x_{j-1}

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $\Omega_j = \texttt{PrincipalSupport}_s(r_j) \cup T_{j-1}$ intermediate expanded subspace
- $w_j = \underset{\text{supp}(z)\subset\Omega_j}{\arg\min} \|y - \mathcal{A}z\|_2$ optimal approximation supported on Ω_j
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

CoSaMP

A different, two-stage approach.

Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support set of x_{j-1}

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $\Omega_j = \texttt{PrincipalSupport}_s(r_j) \cup T_{j-1}$ intermediate expanded subspace
- $w_j = \arg\min_{\text{supp}(z)\subset\Omega_j} \|y - \mathcal{A}z\|_2$ optimal approximation supported on Ω_j
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

CoSaMP

A different, two-stage approach.

Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support set of x_{j-1}

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $\Omega_j = \texttt{PrincipalSupport}_s(r_j) \cup T_{j-1}$ intermediate expanded subspace
- $w_j = \underset{\text{supp}(z)\subset\Omega_j}{\arg\min} \|y - \mathcal{A}z\|_2$ optimal approximation supported on Ω_j
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

CoSaMP

A different, two-stage approach.

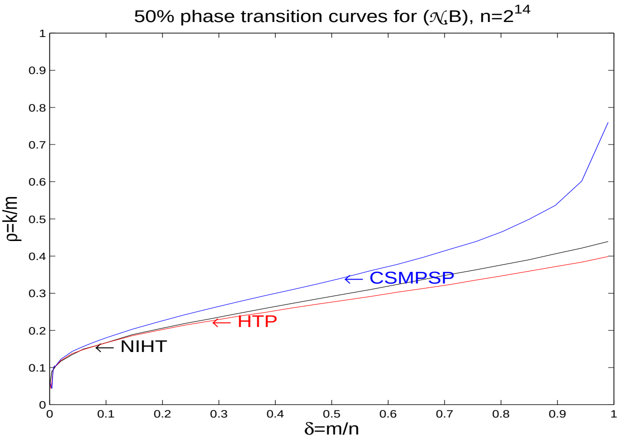
Previous Iteration

- x_{j-1} a s-sparse approximation
- T_{j-1} the support set of x_{j-1}

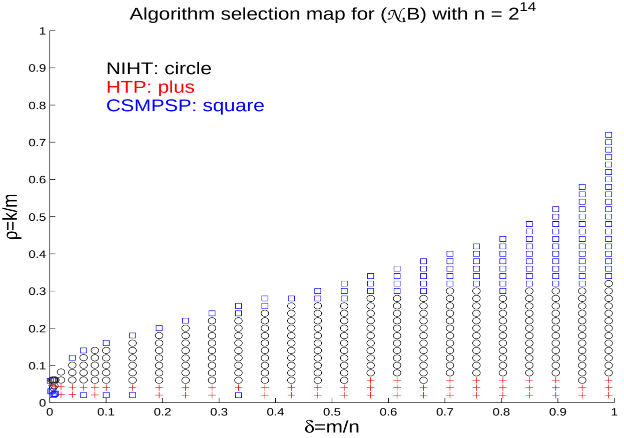
Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- $\Omega_j = \texttt{PrincipalSupport}_s(r_j) \cup T_{j-1}$ intermediate expanded subspace
- $w_j = \underset{\text{supp}(z)\subset\Omega_j}{\arg\min} \|y - \mathcal{A}z\|_2$ optimal approximation supported on Ω_j
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the s largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

Performance Comparisons: recovery phase transitions



Performance Comparisons: algorithm selection maps



What did we learn?

- Why is NIHT ever faster than HTP or CSMPSP?
  - NIHT finds the correct support with less computational expenditure.
  - When the support is correct, HTP and CSMPSP have highly advantageous convergence rates.
  - When the support is incorrect, the CG projection incorporates unnecessary computation.
- Can we combine the advantages of all the algorithms into one algorithm?

What did we learn?

- Why is NIHT ever faster than HTP or CSMPSP?
  - NIHT finds the correct support with less computational expenditure.
  - When the support is correct, HTP and CSMPSP have highly advantageous convergence rates.
  - When the support is incorrect, the CG projection incorporates unnecessary computation.
- Can we combine the advantages of all the algorithms into one algorithm?

What did we learn?

- Why is NIHT ever faster than HTP or CSMPSP?
  - NIHT finds the correct support with less computational expenditure.
  - When the support is correct, HTP and CSMPSP have highly advantageous convergence rates.
  - When the support is incorrect, the CG projection incorporates unnecessary computation.
- Can we combine the advantages of all the algorithms into one algorithm?

NIHT

A subspace restricted steepest descent with evolving support.

Previous Iteration

- x_{j-1} a k-sparse approximation

Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- α_j a near optimal step size
- $w_j = x_{j-1} + \alpha_j r_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the k largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

CGIHT

A subspace restricted conjugate gradient search with evolving support.

Previous Iteration

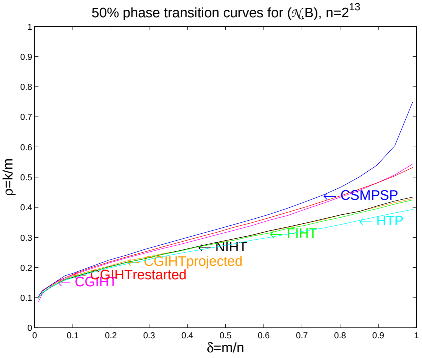
- x_{j-1} a k-sparse approximation
- p_{j-1} the previous search direction

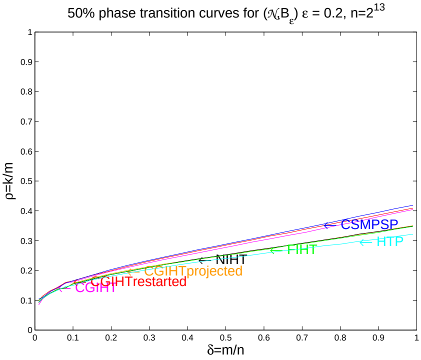
Current Iteration

- $r_j = \mathcal{A}^*(y - \mathcal{A}x_{j-1})$ the steepest descent direction
- β_j a conjugate orthogonalization weight
- $p_j = r_j + \beta_j p_{j-1}$ a conjugate search direction
- α_j a near optimal step size
- $w_j = x_{j-1} + \alpha_j p_j$ an updated approximation (not sparse)
- $T_j = \texttt{PrincipalSupport}_s(w_j)$ the support set of the k largest magnitude entries in w_j
- $x_j = \texttt{Threshold}(w_j, T_j)$ retains the values on T_j and zeros all other entries (hard thresholding)

Performance Comparisons: recovery phase transitions

Compressed Sensing



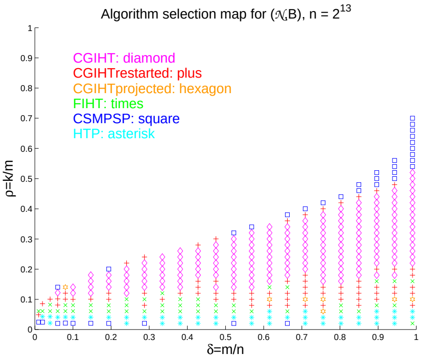


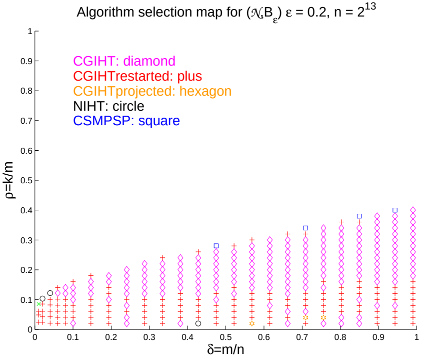
Left: $y = \mathcal{A}x$.

Right: $y = \mathcal{A}x + e$ with $\|e\|_2 = \epsilon\|y\|_2$.

Performance Comparisons: recovery phase transitions

Compressed Sensing



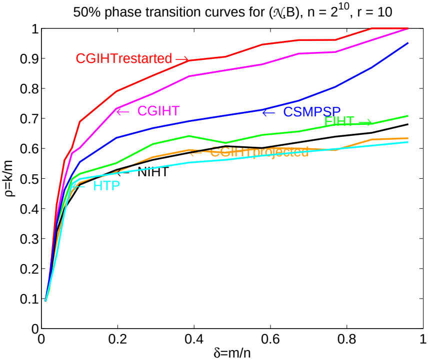


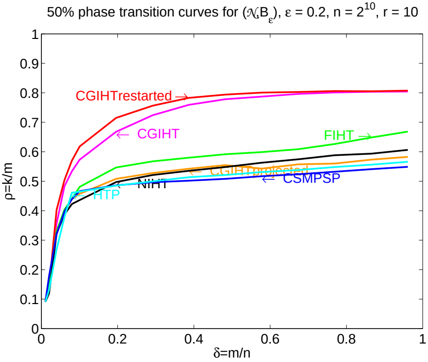
Left: $y = \mathcal{A}x$.

Right: $y = \mathcal{A}x + e$ with $\|e\|_2 = \epsilon\|y\|_2$.

Performance Comparisons: recovery phase transitions

Row Sparse Approximation



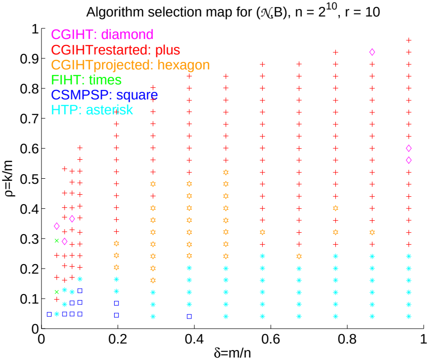


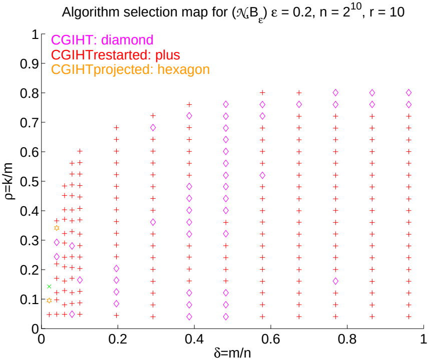
Left: $Y = \mathcal{A}X$.

Right: $Y = \mathcal{A}X + E$ with $\|E\|_F = \epsilon \|Y\|_F$.

Performance Comparisons: algorithm selection maps

Row Sparse Approximation



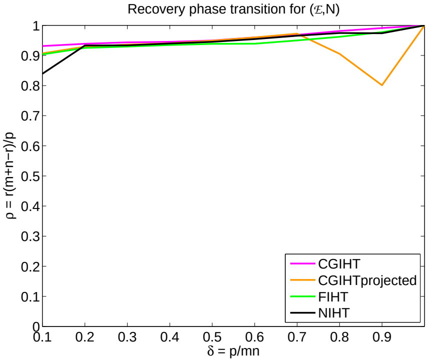


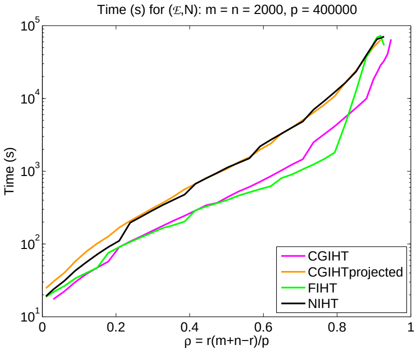
Left: $Y = \mathcal{A}X$.

Right: $Y = \mathcal{A}X + E$ with $\|E\|_F = \epsilon \|Y\|_F$.

Performance Comparisons: matrix completion

Matrix Completion





$y = \mathcal{A}(X)$ with $\mathcal{A}$ entry sensing. Left: recovery phase transitions. Right: average recovery time.

GAGA

GAGA: GPU Accelerated Greedy Algorithms for Compressed Sensing

with Jared Tanner (Oxford)

www.gaga4cs.org

- Fast GPU implementations of greedy algorithms executed from Matlab.
- DCT, Sparse, Dense sensing matrices.
- Several random vector ensembles.
- NIHT, HTP, CSMPSP, CGIHT, $\ldots$
- Solve problems up to 2^{20} in fractions of a second. $(40\times - 60\times$ acceleration)
- Robust testing suite.
- Freely available for research.
- Extension to matrix completion in progress.
- Requires CUDA capable NVIDIA GPU.
- Does NOT require parallel processing toolbox.

GAGA

GAGA: GPU Accelerated Greedy Algorithms
for Compressed Sensing
with Jared Tanner (Oxford)
www.gaga4cs.org

- Fast GPU implementations of greedy algorithms executed from Matlab.
- DCT, Sparse, Dense sensing matrices.
- Several random vector ensembles.
- NIHT, HTP, CSMPSP, CGIHT, ...
- Solve problems up to 2^{20} in fractions of a second. $(40\times - 60\times$ acceleration)
- Robust testing suite.
- Freely available for research.
- Extension to matrix completion in progress.
- Requires CUDA capable NVIDIA GPU.
- Does NOT require parallel processing toolbox.

References

- www.gaga4cs.org
- www.math.grinnell.edu/~blanchaj

- *GPU Accelerated Greedy Algorithms for Compressed Sensing*. Blanchard & Tanner, Math. Prog. Comp., 5(3): 267-304, 2013
- *Performance Comparisons of Greedy Algorithms for Compressed Sensing*. Blanchard & Tanner, Num. Lin. Alg. App., 22(2): 254-282, 2015
- *Conjugate Gradient Iterative Hard Thresholding for compressed sensing and matrix completion*. Blanchard, Tanner, & Wei, Information and Inference, 4(4): 289-327, 2015
- *Conjugate Gradient Iterative Hard Thresholding: observed noise stability for compressed sensing*. Blanchard, Tanner, & Wei, IEEE Trans. Sig. Proc., 63(2): 528-537, 2015

References

- www.gaga4cs.org
- www.math.grinnell.edu/∼blanchaj
- *GPU Accelerated Greedy Algorithms for Compressed Sensing*. Blanchard & Tanner, Math. Prog. Comp., 5(3): 267-304, 2013
- *Performance Comparisons of Greedy Algorithms for Compressed Sensing*. Blanchard & Tanner, Num. Lin. Alg. App., 22(2): 254-282, 2015
- *Conjugate Gradient Iterative Hard Thresholding for compressed sensing and matrix completion*. Blanchard, Tanner, & Wei, Information and Inference, 4(4): 289-327, 2015
- *Conjugate Gradient Iterative Hard Thresholding: observed noise stability for compressed sensing*. Blanchard, Tanner, & Wei, IEEE Trans. Sig. Proc., 63(2): 528-537, 2015